Universidade de Brasília - Instituto de Ciências Exatas
Departamento de Ciência da Computação

Segurança Computacional – Trabalho 1

Cifra de Vigenère

Prof. João Gondim
agosto de 2021

Alunos:
Vitor Vasconcelos de Oliveira
Álvaro Veloso Cavalcanti Luz

Matrículas:
180114778
180115391

O primeiro trabalho da disciplina de segurança computacional consiste em duas partes:

- Criação de um cifrador/decifrador de Vigenère;
- O ataque de recuperação de senha por análise de frequência;

O trabalho foi produzido em Python.

1. Considerações iniciais:

Primeiramente, é válido mencionar que para este programa apenas são permitidas chaves de codificação com tamanho inferior ou igual a 20 caracteres. Tal limitação foi imposta para limitar o número de testes para o tamanho da senha ao executar a tentativa de quebra de senha.

Ademais, houveram dificuldades ao executar os testes impostos pelo professor. No segundo teste passado, o em português, foi obtida a resposta “selporal’ em vez de “temporal”. Apesar disso, em outros testes feitos durante o desenvolvimento do projeto, o código foi capaz de encontrar a decodificação correta, segue um exemplo:

TEXTO CODIFICADO:
eozprdsovibuqeqrbijdbbeatiyfbipsrghneoruytjmbizpfrndbreoornsjlrlrsvbvunourbn
benersgewenfeeotrdbpbifagerunnqopcbreevotoypfqhevnttnlbubrrphbmipafehuv
nqobsgrndjcbefpprgutufsnsrrfavsbhfrqevrpdbteoooeepecehvnrjofnbmfsnfvmeeu
ozeoatenrtehsnvpsfaatpsraajpsfehnpmrcbmqlrtbesaceqrpdraycbngaeakonopasl

bsyeppblqotayvndproioibnbfeaocvspoyaiirreecahlblropaeibmvgveygnbsirleagarlt
oozngndfbeataoarbbusbbn
SENHA:
banana
TEXTO ORIGINAL:
dompedroiioupedroiidobrasilfoiosegundoeultimoimperadordobrasilelesubiuaotro
noemeesteveafrentedopaisatequandoocorreuogolpequeinstalouarepublicasegui
ndoastradicoesportuguesasereaisoherdeirodotronorecebeuvariosnomesafimde
homenagearseusavossantoseanjosseunomecompletoerapedrodealcantarajoao
carlosleopoldosalvadorbibianofranciscoxavierdepaulaleocadiomiguelgabrielrafa
elgonzagadebraganaebourbon

Apesar destas complicações, o resto do projeto apresenta os resultados corretos, cumprindo todos os pré-requisitos impostos na especificação.

2. Criação de um cifrador/decifrador de Vigenère:

A primeira parte do trabalho consiste em duas principais funções:

- **cifra(texto,senha):** A função responsável por cifrar a mensagem. De maneira simples, nela encontramos para cada caractere da frase e da senha suas posições no alfabeto, após isso realizamos a soma de suas posições correspondentes no alfabeto para encontrarmos o caractere correto na cifra. (Obs: caso a posição resultante da soma ultrapasse as 26 letras do alfabeto, simplesmente continuamos a contagem do início novamente). (Obs2: quando a senha for menor que a frase, com a ajuda de um contador, damos voltas na senha, relendo-a do começo até que o texto chegue ao fim)
- **decifra(texto,senha):** A função responsável por decifrar a mensagem. O mesmo processo é realizado na função de cifrar, porém subtraímos as posições ao invés de somá-las.

```python
def cifra(frase, senha):
    new_frase = ''
    frase = frase.lower()
    senha = senha.replace(" ", "")
    contS = 0
    for i in range(0, len(frase)):
        if frase[i] in alfabeto:
            x = alfabeto.find(frase[i])
            if(contS == len(senha)):
                contS = 0
            y = alfabeto.find(senha[contS])
            if(x+y <= 25):
                new_frase = new_frase + alfabeto[x+y]
            else:
                new_frase = new_frase + alfabeto[x+y-26]
            contS += 1
        else:
            new_frase = new_frase + ""
    return new_frase.upper()
```

```python
def decifra(frase, senha):
    new_frase = ''
    frase = frase.lower()
    senha = senha.replace(" ", "")
    contS = 0
    for i in range(0, len(frase)):
        if frase[i] in alfabeto:
            x = alfabeto.find(frase[i])
            if(contS == len(senha)):
                contS = 0
            y = alfabeto.find(senha[contS])
            if(x+y >= 0):
                new_frase = new_frase + alfabeto[x-y]
            else:
                new_frase = new_frase + alfabeto[x-y+26]
            contS += 1
        else:
            new_frase = new_frase + ""
    return new_frase.upper()
```

3. Ataque de recuperação de senha por análise de frequência:

A quebra da senha foi feita utilizando-se de duas análises descritas a seguir.

-Análise para o tamanho da senha:

Para esta etapa, busca-se um padrão de repetição no texto codificado utilizando-se de uma análise do índice de coincidência de sequências de letras com um espaço *n* entre si, sendo *n* um dos valores possíveis para o tamanho da senha. Utiliza-se então dos valores de índice calculados para estas sequências para montar-se uma tabela ordenada. É dado como tamanho correto aquele que não é múltiplo de outro tamanho na tabela e que possui o maior valor de índice.

Segue o código-fonte referente às funções get_tamanho_senha e get_indice, que correspondem à implementação do trecho descrito, sendo get_indice a função que realiza o cálculo do índice de coincidência e get_tamanho_senha a função responsável por montar a tabela mencionada e retornar o valor mais provável para a senha.

```python
def get_tamanho_senha(texto):
    tabela_indice = []
    # Quebra o texto cifrado em sequencias baseadas no comprimento de chave de 0 ao tamanho maximo.
    for tamanho in range(MAX_SENHA):
        # A chave com maior IC eh a chave mais provavel
        soma_indice = 0.0
        media_indice = 0.0
        for i in range(tamanho):
            sequencia = ""
            for j in range(0, len(texto[i:]), tamanho):
                sequencia += texto[i+j]
            if len(sequencia) > 1:
                soma_indice += get_indice(sequencia)
        # se o tamanho for diferente de 0
        if not tamanho == 0:
            media_indice = soma_indice/tamanho

        tabela_indice.append(media_indice)

    # retorna o comprimento da chave com  maior indice de coincidencia (chave mais provavel)
    melhor_tamanho = tabela_indice.index(sorted(tabela_indice, reverse=True)[0])
    segundo_melhor_tamanho = tabela_indice.index(sorted(tabela_indice, reverse=True)[1])

    if melhor_tamanho % segundo_melhor_tamanho == 0:
        # se sao multiplos
        return segundo_melhor_tamanho
    else:
        # se nao sao
        return melhor_tamanho
```

```python
# Achando o indice de coincidencia atraves da formula
def get_indice(texto):
    N = float(len(texto))
    soma_frequencias = 0.0

    # Usinando a formula do indice de coincidencia
    for letra in alfabeto:
        soma_frequencias += texto.count(letra) * (texto.count(letra)-1)

    indice = soma_frequencias/(N*(N-1))

    return indice
```

-Análise de deslocamento utilizando qui-quadrados:

Após ter-se deduzido o tamanho da chave, o processo de decodificação é seguido por uma etapa de análise utilizado cálculo estatístico. Para isso, a frase original é dividida sequências de letras com um espaço *K* entre si, sendo *K* o valor correspondente ao tamanho deduzido anteriormente para a chave. Utiliza-se então o modelo qui-quadrados para testar qual letra do alfabeto produz um offset na sequência que mais se assemelha à distribuição de probabilidades de cada letra na língua selecionada.

Segue o código-fonte referente à implementação do trecho descrito. A função get_senha separa as sequências anteriormente mencionadas do texto codificado e redireciona para a função analisa_freq, que realiza o cálculo de qui-quadrados e retorna a letra que mais provavelmente foi utilizada para codificar a sequência em questão. Depois de obtida a sequência de letras da chave, get_senha retorna a sequência completa.

```python
# Performa uma analise de frequencia no texto e retorna a letra decodificada para determinada parte da chave
# Usa o calculo estatistico de qui-quadrado para testar o quao similar duas distribuicoies sao
def analisa_freq(sequencia):
    qui_quadrados = [0] * len(alfabeto)

    for i in range(len(alfabeto)):
        soma_quadrados = 0.0
        offset = []
        valor_obs = [0] * len(alfabeto)

        for j in range(len(sequencia)):
            offset.append(
                chr(((ord(sequencia[j])-ord('a')-i) % len(alfabeto))+ord('a')))

        # conta o numero de vezes que cada letra do offset aparece para o calculo de frequencia
        for j in offset:
            valor_obs[ord(j) - ord('a')] += 1

        # divide o os valores de obs pelo tamanho da sequencia
        for j in range(len(alfabeto)):
            valor_obs[j] = valor_obs[j]/float(len(sequencia))

        # comparando utilizando a formula do qui-quadrado
        for j in range(len(alfabeto)):
            if(idioma == 1):
                soma_quadrados += ((valor_obs[j] - float(frequencias_PTBR[j])) * (
                    valor_obs[j] - float(frequencias_PTBR[j])))/float(frequencias_PTBR[j])
            elif(idioma == 2):
                soma_quadrados += ((valor_obs[j] - float(frequencias_ENG[j])) * (
                    valor_obs[j] - float(frequencias_ENG[j])))/float(frequencias_ENG[j])
        # adiciona na tabela
        qui_quadrados[i] = soma_quadrados

    # menor valor equivale ao deslocamento equivalente ao letra da chave
    deslocamento = qui_quadrados.index(min(qui_quadrados))

    # retorna a letra para qual a sequencia foi deslocada
    return chr(deslocamento+ord('a'))
```

```python
# funcao responsavel por descobrir e retornar a senha
# realiza isso analisando a frequencia de cada letra da senha apra varios sequencias do texto
def get_senha(texto, tamanho):
    senha = ''
    for i in range(tamanho):
        sequencia = ""
        for j in range(0, len(texto[i:]), tamanho):
            sequencia += texto[i+j]
        senha += analisa_freq(sequencia)

    return senha
```